

# *ACTIONCODE CHATBOT*

## **Project by:**

<i>Muhammad Zain Farooq Khan</i>	<i>2974976</i>
<i>Farrukh Jahangeer</i>	<i>2960928</i>
<i>Khusmanda Ramanjooloo</i>	<i>2953316</i>

## Table of Contents

<b>ABSTRACT</b> .....	3
INTRODUCTION .....	3
BACKGROUND .....	4
COLLABORATION .....	5
FEATURE .....	5
MILESTONE 1 .....	6
MILESTONE 2 .....	11
MILESTONE 3 .....	14
weatherCheck () .....	16
cloth () Method .....	17
Running chatbot .....	22
Converting Kelvin to degree Celsius .....	24
Implementing an Array of random type of clothing .....	28
JUnit test suite .....	34
DESIGN .....	37
RUNNING THE CHATBOT .....	38
FUTURE ADDITION .....	38
CONCLUSION .....	38

## ABSTRACT

ActionCode is a custom chatbot by AIML which our team is using to develop a bit further. ActionCode is being developed by a team of three person Muhammad Zain Farooq Khan, Farrukh Jahangeer and Khusmanda Ramanjooloo. ActionCode has been created during a period of six weeks.

ActionCode is constructed to search for weather condition for specific places and suggest some clothes we can pack for that particular place based on weather condition.

ActionCode has four particular part:

- \* Normal chat with chatbot
- \* Finding the weather at a particular
- \* Suggesting clothes that can be pack for the trip

ActionCode uses an external API which is the OpenWeatherMap API. ActionCode contains the weatherC java file which implements the External API methods. It contains a weatherCTest java file which does Junit testing.

## INTRODUCTION

The chatbot ActionCode has been constructed using AIML custom chatbot and Open Weather Map API to find the weather, it is constructed by a team of three namely Muhammad Zain Farooq Khan, Farrukh Jahangeer and Khusmanda Ramanjooloo.

ActionCode is made up of one java file weatherC that consist of nine methods namely weather Check, cloth, conversionMin, conversionMax, zero, aboveZero, aboveTen, aboveFifteen, aboveEighteen.

## BACKGROUND

ActionCode is constructed on the basis of AIML Chatbot. AIML Chatbot, Artificial Intelligence Markup Language, is an XML-based language used to program natural language software agents, such as chatbots. Richard Wallace originally developed AIML. The team prepare the chatbot by coding it how to find the weather using Open Weather Map.

First of all, we downloaded the AIML Custom Chatbot (Java chatbot example using aiml library, n.d.) in our laptop and open it in eclipse Maven project. We added the Ab.jar in the library and the dependency in the pom.xml file. We Copied the bot folder from program-ab directory into the resources folder the maven project. This folder contains default AIML sets that we will use initially. We then run the AddAiml.java file with java application, followed by running the Chatbot.java file.

ActionCode was initially designed to run through Google Assistance but due to some difficulties we implemented the API directly in the chatbot. The OpenWeatherMap API is an online service that provides weather data. We sign up for the OpenWeatherMap, we got an API key that we need to put into our java code in order to obtain the weather.

The objectives of ActionCode is divided into three milestones.

### **Objective for first Milestone:**

- \* Download the AIML chatbot
- \* Implement changes and ask chatbot questions
- \* Work on Google Assistance – try to implement location

### **Objective for Second Milestone:**

- \* Enter location of certain places
- \* Find weather

### **Objective for third Milestone:**

- \* Link weather, location with chatbot
- \* Test for few locations at once
- \* Check for appropriate weather

## COLLABORATION

To make the ActionCode into a success, we used TeamViewer to share our work and collaborate during these six weeks. Everyone was assigned a part of the code to complete. While coding each part, the code was shared so everyone is at same working level.

Twice a week, team members were having meeting to see how the ActionCode is being created, how much work has been done in constructing it and what new features should be implemented to make the chatbot overcome its weaknesses.

When a member was having difficulties, it was solved in group by arranging for a quick meeting. It is a very collaborating team.

## FEATURE

ActionCode is designed with these features:

- The ability to answer questions and communicate about things that it has been programmed with.
- Enables User to look for weather condition by entering city name.
- Weather information of five different locations can be found at once.
- The minimum and maximum temperature is given in degree Celsius.
- Relevant suggestion of cloth depending on minimum temperature at that location.
- Random selection of clothing from array implemented.
- Different temperature has different array.

## MILESTONE 1

### PLAN FOR FIRST MILESTONE:

- \* Download the AIML chatbot
- \* Implement changes and ask chatbot questions
- \* Work on Google Assistance

### IMPLEMENTATION:

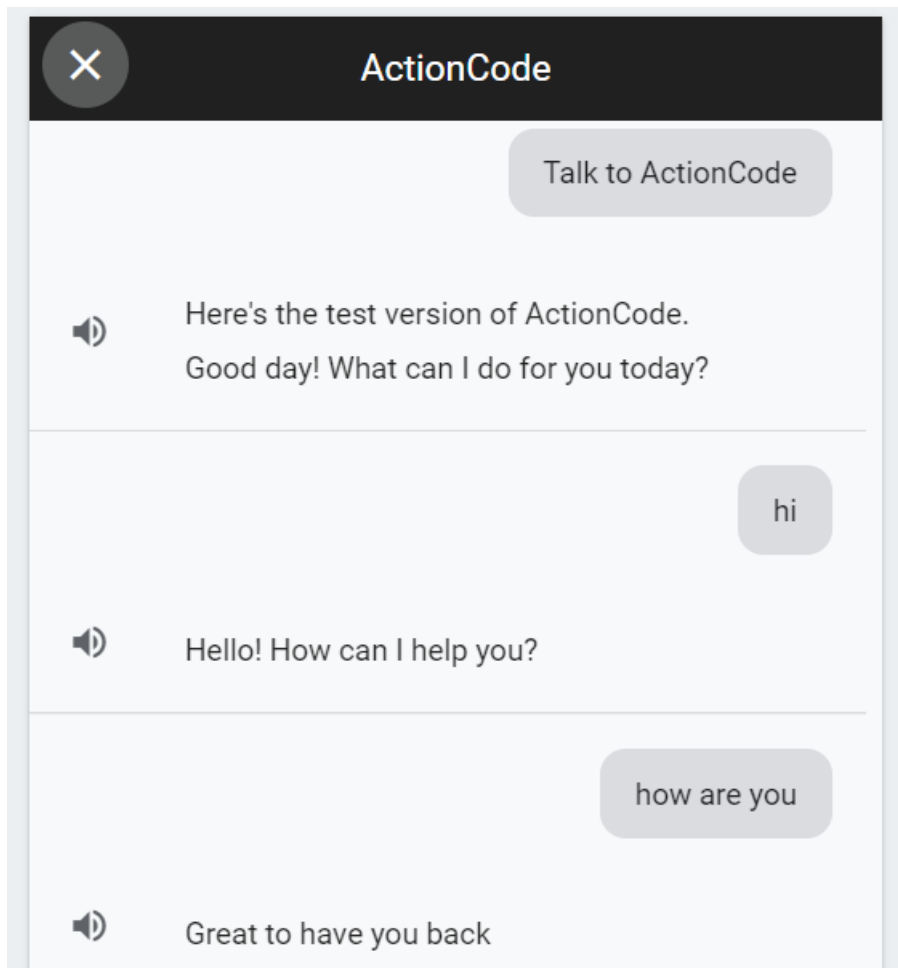
- \* A new AIML file is made which is called weatherCloth.  
Where the user says the weather and ActionCode suggest a clothing type.
- \* Create an agent in Google Assistance – called ActionCode

The AIML weatherCloth file picks a random output when a certain weather is entered just in case the user will want to pack clothes in a different way. For one weather condition, it is put in a category, which is followed by a pattern that the human user would enter. Afterward the template comes which are answers that ActionCode will be using give back.

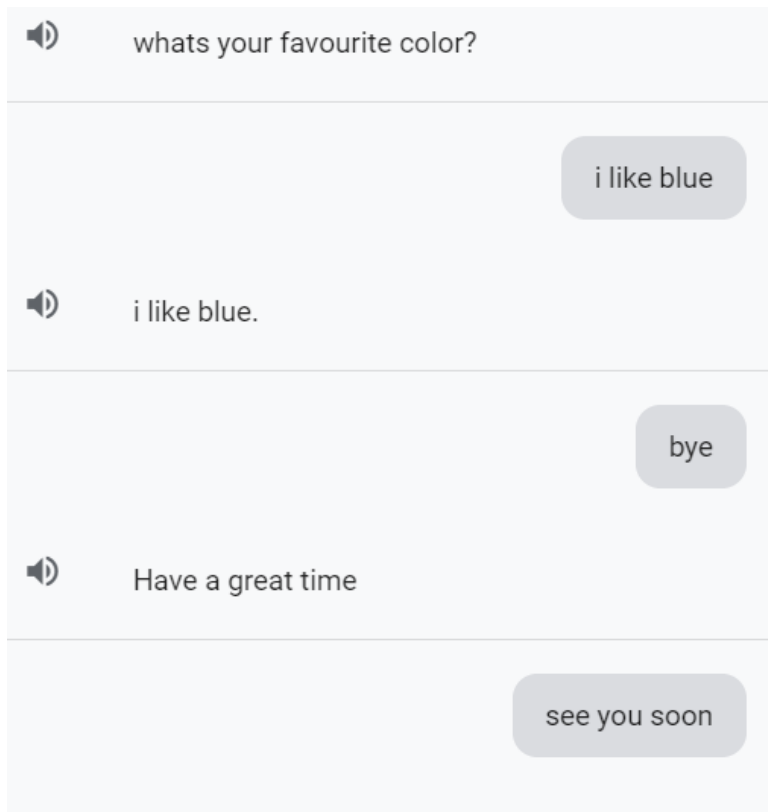
```
1<?xml version="1.0" encoding="UTF-8"?>
2<aiml>
3<!-- File: WeatherCloth.aiml -->
4<!-- Author: Dr. Richard S. Wallace -->
5<!-- Last modified: April 22, 2020 -->
6<!-- -->
7<!-- This AIML file is part of the SUPER 0.0.4 chat bot knowledge base. -->
8<!-- -->
9<!-- The SUPER brain is Copyright &copy; 2020 by ALICE AI Foundation. -->
10<!-- -->
11<!-- The SUPER brain is released under the terms of the GNU Lesser General
12<!-- Public License, as published by the Free Software Foundation. -->
13<!-- -->
14<!-- This file is distributed WITHOUT ANY WARRANTY; without even the -->
15<!-- implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE
16<!-- -->
17<!-- For more information see http://www.alicebot.org -->
18<category><pattern>ITS SUNNY</pattern>
19<template><random>
20<li>Pack clothes some light clothes and take sun cream.</li>
21<li>Pack some short sleeve clothes.</li>
22<li>Pack some light clothes and sunglasses</li>
23</random></template>
24</category>
25<category><pattern>ITS RAINNY</pattern>
26<template><random>
27<li>Pack warm clothing and take umbrella.</li>
28<li>Take long sleeve clothes to keep warm.</li>
29<li>Take warm clothes and a raincoat.</li></random></template>
30</category>
31<category><pattern>ITS HAILING</pattern>
```

## TESTING:

Test with ActionCode – Google Assistance







## Test with Chatbot

```
Human : HI
Robot : Hi it's great to see you!
Human : HOW ARE YOU
Robot : Glad to see you.
Human : WHAT IS IRELAND
Robot : It's wonderful country.
Human : WHATS YOUR JOB
Robot : I'm a mobile virtual assistant.
Human : WHATS YOUR HOBBIES
Robot : I'm dedicated to human transformation.
Human :
```

## Testing the weatherCloth AIML file

```
Robot : Today is April 07, 2020
Human : its sunny
Robot : Pack some light clothes and sunglasses
Human : its cloudy
Robot : Pack some normal wear the temperature is mild.
Human : its rainy
Robot : Take long sleeve clothes to keep warm.
Human : its hailing
Robot : Take warm clothes.
Human : its snowing
Robot : Take long sleeve clothes to keep warm.
Human : its snowing
Robot : Don't forget your Jacket and boots.
Human : its rainy
Robot : Take warm clothes and a raincoat.
Human :
```

## UPDATE:

These are the plan for the next milestone review.

- \* Implement weather in Google Assistance
- \* Develop ActionCode to take in location of places
- \* Link ActionCode and Google Assistance

## MILESTONE 2

### PLAN FOR SECOND MILESTONE:

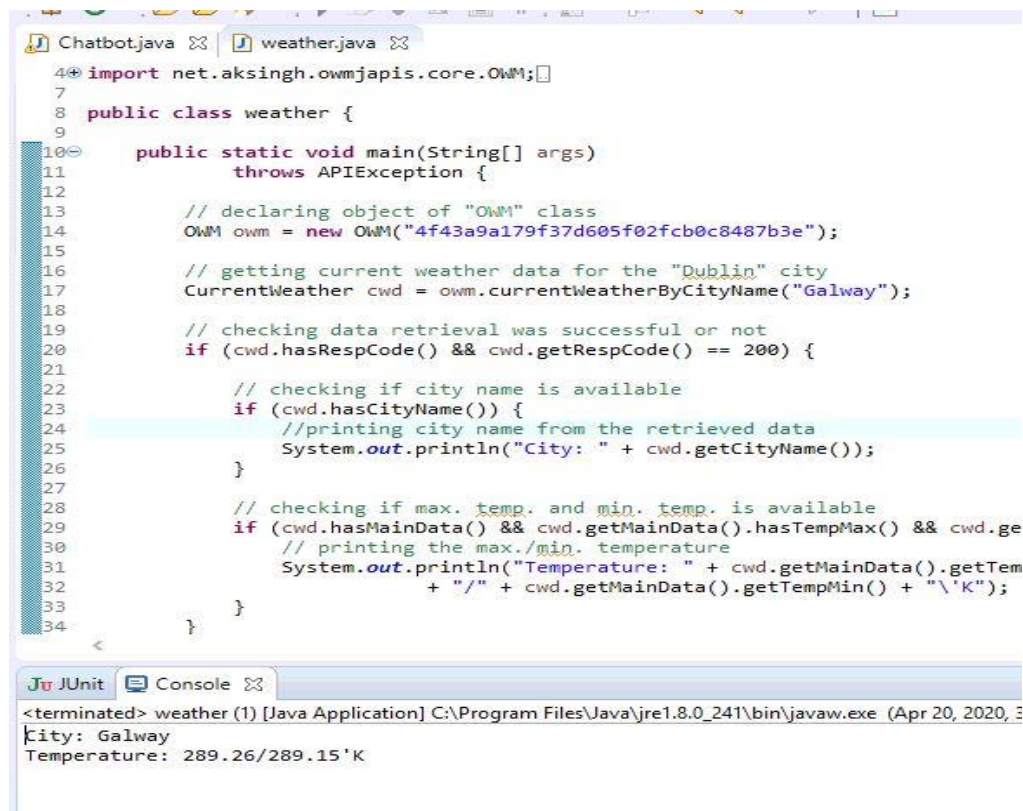
- \* Make java file for Weather API
- \* Connect Chatbot with API

### IMPLEMENTATION:

- \* A java file has been created called WeatherC.java
- \* WeatherC.java provides current minimum and maximum temperature in Kelvin.
- \* WeatherC takes a location from user and output the temperature.

WeatherC java file is created in the src/main/java inside the package.

- The key for the API is obtain by signing up for OpenWeatherMap.
- Download the OWM jar files that would enable us to run the OpenWeatherMap from Chatbot. The jar files are stored in the library of the chatbot and the path has to be built for it.
- The API key is used in the code while declaring an object for the OWM.
- Will be searching for the weather according to the name of the city.
- Finally getting the minimum and maximum temperature in kelvin.



```
Chatbot.java weather.java
4 import net.aksingh.owmjapis.core.OWM;
7
8 public class weather {
9
10     public static void main(String[] args)
11         throws APIException {
12
13         // declaring object of "OWM" class
14         OWM owm = new OWM("4f43a9a179f37d605f02fcb0c8487b3e");
15
16         // getting current weather data for the "Dublin" city
17         CurrentWeather cwd = owm.currentWeatherByCityName("Galway");
18
19         // checking data retrieval was successful or not
20         if (cwd.hasRespCode() && cwd.getRespCode() == 200) {
21
22             // checking if city name is available
23             if (cwd.hasCityName()) {
24                 //printing city name from the retrieved data
25                 System.out.println("City: " + cwd.getCityName());
26             }
27
28             // checking if max. temp. and min. temp. is available
29             if (cwd.hasMainData() && cwd.getMainData().hasTempMax() && cwd.ge
30                 // printing the max./min. temperature
31                 System.out.println("Temperature: " + cwd.getMainData().getTem
32                     + "/" + cwd.getMainData().getTempMin() + "\"K");
33             }
34         }
35     }
36 }
```

JUnit Console

```
<terminated> weather (1) [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (Apr 20, 2020, 3
City: Galway
Temperature: 289.26/289.15'K
```

Running this piece of code gives as output the name of the city and the min/max temperature in kelvin.

Next implementation is taking user input.

- To do so we import scanner library.
- Assign the user input as kb.
- Write an output statement that ask the user to input a location.
- Store the location the user has entered.
- Pass the location in the object that will be finding the weather.

Ask user for location

```
1 package com;
2 import java.io.File;
10
11 public class weatherC {
12     private static final boolean TRACE_MODE = false;
13     static String botName = "super";
14
15     public static void main(String[] args)
16         throws APIException {
17
18         Scanner kb = new Scanner(System.in);
19
20         //Ask user for location
21         System.out.println("Enter a location: ");
22         String location = kb.nextLine();
23
24         String resourcesPath = getResourcesPath();
25         System.out.println(resourcesPath);
26         MagicBooleans.trace_mode = TRACE_MODE;
27         Bot bot = new Bot("super", resourcesPath);
```

Console | Git Staging | Properties | JUnit | Coverage | Synchronize

weatherC [Java Application] C:\Program Files\Java\jdk-11\bin\javaw.exe (21 Apr 2020, 16:44:02)

Enter a location: |

Location entered: Dublin

```
1 package com;
2 import java.io.File;
10
11 public class weatherC {
12     private static final boolean TRACE_MODE = false;
13     static String botName = "super";
14
15     public static void main(String[] args)
16         throws APIException {
17
18         Scanner kb = new Scanner(System.in);
19
20         //Ask user for location
21         System.out.println("Enter a location: ");
22         String location = kb.nextLine();
23
24         String resourcesPath = getResourcesPath();
25         System.out.println(resourcesPath);
26         MagicBooleans.trace_mode = TRACE_MODE;
27         Bot bot = new Bot("super", resourcesPath);
```

Console Git Staging Properties JUnit Coverage Synchronize  
weatherC [Java Application] C:\Program Files\Java\jdk-11\bin\javaw.exe (21 Apr 2020, 16:45:28)  
Enter a location:  
Dublin

## Testing of temperature

Location: Dublin

```
46         if (cwd.hasMainData() && cwd.getMainData())
47             // printing the max./min. temperature
48             System.out.println("Temperature: " + c
49                 + "/" + cwd.getMainData()).
50         }
51     }
52 }
53 private static String getResourcesPath() {
```

Console Git Staging Properties JUnit Coverage Syr  
<terminated> weatherC [Java Application] C:\Program Files\Java\jdk-11\bin\javaw  
update.aiml  
utilities.aiml  
WeatherCloth.aiml  
weatherDetail.aiml  
Loaded 4851 categories in 2.862 sec  
writeAIMLIFFiles  
--> Bot super 4851 completed 0 deleted 0 unfinished  
City: Dublin  
Temperature: 286.15/283.15'K

## UPDATE

- \* Connect the weatherC.java result with a suggestion of clothes to be pack
- \* Apply proper connection with chatbot and weather API

## MILESTONE 3

### PLANS FOR THIRD MILESTONE:

- \* Connected the weatherC.java result with a suggestion of clothes to be pack.
- \* Apply proper connection with chatbot and weather java File.
- \* Ask user to add 5 different Locations.

### IMPLEMENTATION:

- \* In the WeatherC java file, two methods are added.
- \* A Junit test file is created containing one method.

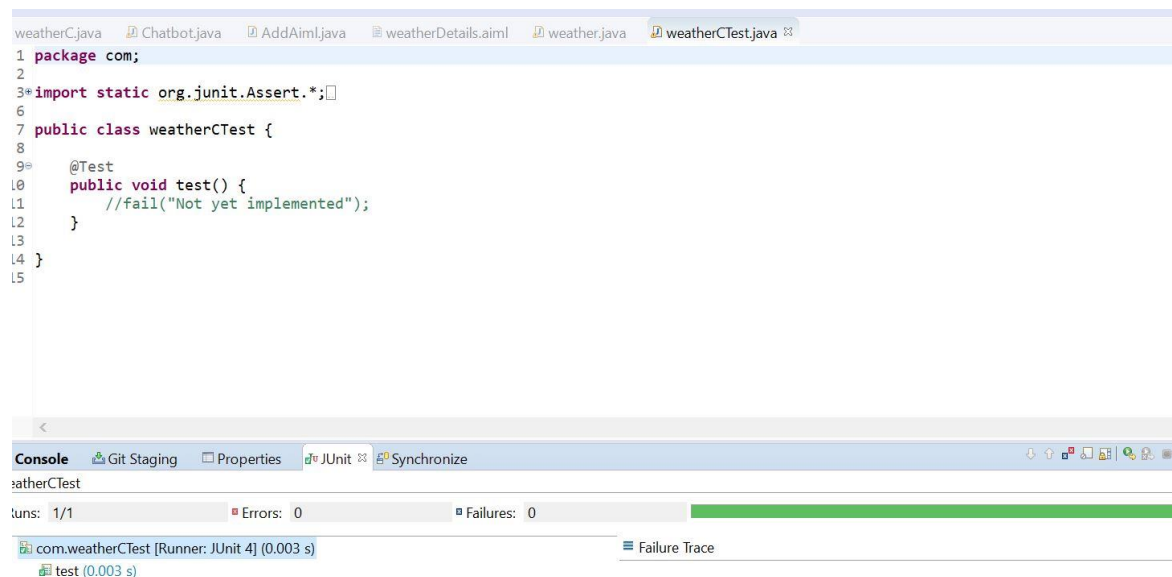
#### Junit weatherCTest

A java file is created called weatherCTest.java, it is a Junit test file.

It consists of one method namely testClothes (). testClothes is a void method.

First of all, when the file is created, we make sure that it gives a Green bar before starting with it.

The first Green bar obtain when file is created.

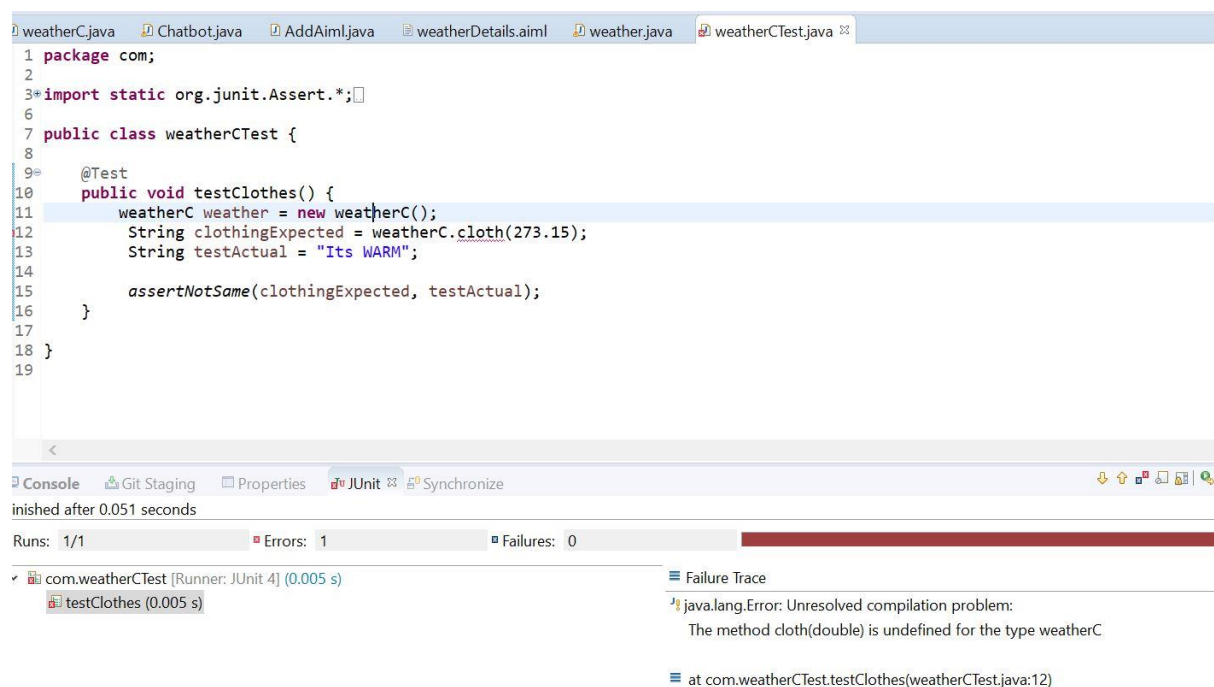


The screenshot shows an IDE with a Java file named `weatherCTest.java` open. The code is as follows:

```
1 package com;
2
3 import static org.junit.Assert.*;
4
5
6
7 public class weatherCTest {
8
9     @Test
10    public void test() {
11        //fail("Not yet implemented");
12    }
13
14 }
15
```

Below the code editor, the **JUnit** tab is active in the **Console** panel. It shows a green progress bar indicating that the test passed. The status bar at the bottom indicates: **Runs: 1/1**, **Errors: 0**, **Failures: 0**. The test output shows: `com.weatherCTest [Runner: JUnit 4] (0.003 s)` and `test (0.003 s)`.

After adding the Junit testing code and running it, the test gives a red bar as the method as not yet exist in main weatherC file.



The screenshot shows the same IDE with the `weatherCTest.java` file. The code is now:

```
1 package com;
2
3 import static org.junit.Assert.*;
4
5
6
7 public class weatherCTest {
8
9     @Test
10    public void testClothes() {
11        weatherC weather = new weatherC();
12        String clothingExpected = weather.cloth(273.15);
13        String testActual = "Its WARM";
14
15        assertNotSame(clothingExpected, testActual);
16    }
17
18 }
19
```

The **JUnit** tab in the **Console** panel now shows a red progress bar, indicating a test failure. The status bar at the bottom indicates: **Runs: 1/1**, **Errors: 1**, **Failures: 0**. The test output shows: `com.weatherCTest [Runner: JUnit 4] (0.005 s)` and `testClothes (0.005 s)`. The **Failure Trace** is expanded, showing the following error:

```
java.lang.Error: Unresolved compilation problem:
  The method cloth(double) is undefined for the type weatherC

    at com.weatherCTest.testClothes(weatherCTest.java:12)
```

After making the cloth () method in weatherC file and running the Junit test.

```
1 package com;
2
3 import static org.junit.Assert.*;
4
5
6
7 public class weatherCTest {
8
9     @Test
10    public void testClothes() {
11        weatherC weather = new weatherC();
12        String clothingExpected = weather.cLoth(273.15);
13        String testActual = "Its WARM";
14
15        assertNotSame(clothingExpected, testActual);
16    }
17
18 }
19
```

Console | Git Staging | Properties | JUnit | Synchronize

inished after 0.049 seconds

Runs: 1/1 | Errors: 0 | Failures: 0

com.weatherCTest [Runner: JUnit 4] (0.000 s) | Failure Trace

### weatherCheck ()

- Connects the weather API to the java file.
- Take 5 different locations from the user. A for loop is used to take in the five location and output the relevant temperature in kelvin.
- Check the temperature from API and output the minimum and maximum temperature.
- Call the method cloth (), whereby the minimum temperature is passed to obtain the clothes information.

The location that the user has input is passed in the object where it accesses the OWM API and a temperature is returned.



## WeatherCheck () method Coding

```
public static void weatherCheck()
    throws APIException{
    Scanner kb = new Scanner(System.in);

    int count = 5;
    for(int i = 0; i<count; i++) { // take in 5 location
        System.out.println("");

        //Ask user for location
        System.out.println("Enter a location: ");
        String location = kb.nextLine();

        // declaring object of "OWM" class
        OWM owm = new OWM("4f43a9a179f37d605f02fcb0c8487b3e");

        // getting current weather data for the city
        CurrentWeather cwd = owm.currentWeatherByCityName(location);

        // checking data retrieval was successful or not
        if (cwd.hasRespCode() && cwd.getRespCode() == 200) {

            // checking if city name is available
            if (cwd.hasCityName()) {
                System.out.println();
                //printing city name from the retrieved data
                System.out.println("City: " + cwd.getCityName());
            }

            // checking if max. temp. and min. temp. is available
            if (cwd.hasMainData() && cwd.getMainData().hasTempMax() && cwd.getMainData().hasTempMin()) {
                // printing the max./min. temperature
                System.out.println("Temperature: " + cwd.getMainData().getTempMax()
                    + "/" + cwd.getMainData().getTempMin() + "°K");
                System.out.println(cloth(cwd.getMainData().getTempMin()));
            }
        }
    }
}
```

## cloth () Method

- Take the minimum temperature as Double from the weatherCheck () method.
- Depending on the temperature it goes into a specific if statement whereby the clothing type is determined.
- Print a statement which cloth type to take.

## Cloth () method coding

```
public static String cloth(Double tempMin) {
    String typeOfClothing = "";
    if(tempMin < 273.15) { // less than 0 degree
        typeOfClothing = "It would be cold, take warm clothes, jackets, scarfs and gloves.";
    }
    else if(tempMin > 273.15 && tempMin <= 283.15 ) { //greater than 0 and less than 10
        typeOfClothing = "It would be cold, take warm clothes and jacket.";
    }
    else if(tempMin > 283.15 && tempMin <= 288.15) { // greater than 10 and less than 16
        typeOfClothing = "It would be mild cold, take normal clothing and jacket.";
    }
    else if(tempMin > 288.15 && tempMin <= 291.15) { // greater than 15 and less than 19
        typeOfClothing = "It would be mild and bit warm, take normal clothing and a light jacket.";
    }
    else if(tempMin >= 291.15) { //more than 18
        typeOfClothing = "It would be very warm, take light clothes.";
    }
    return typeOfClothing;
}
```

In the main method the weatherCheck () method is called.

```
public class weatherC {
    public static void main(String[] args) throws APIException
    {

        weatherCheck();
    }
}
```

## Connecting weatherC java file to Chatbot

- In the main method of Chatbot, a code line is added for connection between chatbot and weatherC.
- If the human input “Let’s check the weather”, it will direct the chatbot toward the weatherC java file to the weatherCheck () method.
- After which the location of the place will be input.

### Code from Chatbot.java

```
else if (textLine.equals("Lets check the weather")) {
    weatherC.weatherCheck();
}
- .
```

## Running the WeatherC java file only

### Trial 1:

#### Testing for countries and cities around Europe

Enter a location:

Dublin

City: Dublin

Temperature: 295.93/287.59'K

It would be mild cold, take normal clothing and jacket.

Enter a location:

London

City: London

Temperature: 293.15/290.15'K

It would be mild and bit warm, take normal clothing and a light jacket.

Enter a location:

greenland

City: Greenland

Temperature: 285.37/282.04'K

It would be cold, take warm clothes and jacket.

Enter a location:

spain

Enter a location:

spain

City: Spain

Temperature: 291.15/289.26'K

It would be mild and bit warm, take normal clothing and a light jacket.

Enter a location:

Italy

City: Italy

Temperature: 298.71/298.15'K

It would be very warm, take light clothes.

## Trial 2:

### Testing for countries and cities around World

Enter a location:

Multan

City: Multan

Temperature: 305.15/305.15'K

It would be very warm, take light clothes.

Enter a location:

Mauritius

City: Mauritius

Temperature: 298.15/298.15'K

It would be very warm, take light clothes.

Enter a location:

New York

City: New York

Temperature: 282.59/281.48'K

It would be cold, take warm clothes and jacket.

Enter a location:

Chicago

City: Chicago

Temperature: 289.15/285.37'K

It would be mild cold, take normal clothing and jacket.

Enter a location:

miami

City: Miami

Temperature: 303.71/302.04'K

It would be very warm, take light clothes.

## Chatbot code with new implementation:

```
public class Chatbot {
    private static final boolean TRACE_MODE = false;
    static String botName = "super";

    public static void main(String[] args) {
        weatherC weathercheck = new weatherC();
        try {

            String resourcesPath = getResourcesPath();
            System.out.println(resourcesPath);
            MagicBooleans.trace_mode = TRACE_MODE;
            Bot bot = new Bot("super", resourcesPath);
            Chat chatSession = new Chat(bot);
            bot.brain.nodeStats();
            String textLine = "";

            while(true) {
                System.out.print("Human : ");
                textLine = IOUtils.readInputTextLine();
                if ((textLine == null) || (textLine.length() < 1))
                    textLine = MagicStrings.null_input;
                if (textLine.equals("q")) {
                    System.exit(0);
                } else if (textLine.equals("exit")) {
                    bot.writeQuit();
                    System.exit(0);
                }
                else if (textLine.equals("Lets check the weather")) {
                    weatherC.weatherCheck();
                }
                else {
                    String request = textLine;
                    if (MagicBooleans.trace_mode)
                        System.out.println("STATE=" + request + ":THAT=" + ((History) chatSession.thatHistory.get(0)).get(0)
                            + ":TOPIC=" + chatSession.predicates.get("topic"));
                    String response = chatSession.multisentenceRespond(request);
                    while (response.contains("<"))
                        response = response.replace("<", "<");
                    while (response.contains(">"))
                        response = response.replace(">", ">");
                    System.out.println("Robot : " + response);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

## Running chatbot

```
Reading Learnf file
AIML Set birdfeature not found.
AIML Set birdfeature not found.
Loaded 4862 categories in 0.395 sec
--> Bot super 4862 completed 0 deleted 0 unfinished
28614 nodes 22356 singletons 4862 leaves 0 shortcuts 1396 n-ary 28613 branches 0.9999651 average branching
Human : hi
Robot : Hi how are you?
Human : i am good
Robot : I wonder if that's a problem for you.
Human : how are you
Robot : Excited to talk with you.
Human : date
CalendarUtils.date: April 26, 2020
Robot : Today is April 26, 2020
Human : time
CalendarUtils.date: 7:03 pm
Robot : The time is 7:03 pm
Human : whats your name
Robot : I am called
SUPER.
Human : what are you
Robot : I'm a mobile virtual assistant.
Human : when is your birthday
Robot : My birthday is October 9.
Human : i am going on a trip
Robot : I wonder if that's a problem for you.
Human :
```

```
28614 nodes 22356 singletons 4862 leaves 0 shortcuts 1396 n-ary 28613 br
Human : hi
Robot : Hi how are you?
Human : i am good
Robot : I wonder if that's a problem for you.
Human : how are you
Robot : Excited to talk with you.
Human : date
CalendarUtils.date: April 26, 2020
Robot : Today is April 26, 2020
Human : time
CalendarUtils.date: 7:03 pm
Robot : The time is 7:03 pm
Human : whats your name
Robot : I am called
SUPER.
Human : what are you
Robot : I'm a mobile virtual assistant.
Human : when is your birthday
Robot : My birthday is October 9.
Human : i am going on a trip
Robot : I wonder if that's a problem for you.
Human : Lets check the weather
```

Human : Lets check the weather

Enter a location:

Dublin

City: Dublin

Temperature: 297.59/288.71'K

It would be mild and bit warm, take normal clothing and a light jacket.

Enter a location:

Belfast

City: Belfast

Temperature: 285.15/283.15'K

It would be cold, take warm clothes and jacket.

Enter a location:

London

City: London

Temperature: 293.15/289.15'K

It would be mild and bit warm, take normal clothing and a light jacket.

Enter a location:

Scotland

City: Scotland

Temperature: 297.59/294.15'K

It would be very warm, take light clothes.

Enter a location:

new zealand

|

City: New Zealand

Temperature: 288.25/288.25'K

It would be mild and bit warm, take normal clothing and a light jacket.

Human :

---

Human : Thanks

Robot : I'm always glad to help.

Human : q

|

## Converting Kelvin to degree Celsius

In Junit weatherCTest file add test methods.

### 1. testMin () method

Junit test give Red Bar as conversionMin () method is not present in weather file.

This method creates an object temperature of weatherC java file. A minExpected variable is assigned as integer which join the conversionMin () method to the weatherC java file to test while passing a double 273.15. Integer minActual is put as zero and assertEquals test if both variables are equal before and after conversion.

The screenshot shows an IDE with a Java file containing a JUnit test method. The code is as follows:

```
22
23 @Test
24 public int testMin() {
25     weatherC temperature = new weatherC();
26     int minExpected = temperature.conversionMin(273.15);
27     int minActual = 0;
28
29     assertEquals(minExpected, minActual);
30 }
```

Below the code, the IDE's console and failure trace are visible. The console shows "Finished after 0.044 seconds" and "Runs: 1/1", "Errors: 1", "Failures: 0". The failure trace shows the following error:

```
java.lang.Exception: No tests found matching [{ExactMatcher:DisplayName=testMin}, {ExactMat
assRequest@fcd6521
```

The failure trace is repeated in a separate window below the main IDE window.



Implementing conversionMin () method in weather file.

```
87 //convert min temperature from to degree
88 public static int conversionMin(Double tempMin) {
89     int minTempInDegree = 0;
90
91     minTempInDegree = (int) (tempMin - 273.15);
92
93     return minTempInDegree;
94 }
95
```

JUnit test for testMin () method gives Green Bar after implementation.



## 2. testMax () method

JUnit test give Red Bar as conversionMax () method is not present in weather file.

```
32 @Test
33 public void testMax() {
34     weatherC temperature = new weatherC();
35     int maxExpected = temperature.conversionMax(273.15);
36     int maxActual = 0;
37
38     assertEquals(maxExpected, maxActual);
39 }
40
41
42
```

Red Bar for testMax () method as conversionMax () does not exist in weatherC



Implementing conversionMax () method in weatherC

```
96 //convert max temperature from to degree
97 public static int conversionMax(Double tempMax) {
98     int maxTempInDegree = 0;
99
100     maxTempInDegree = (int) (tempMax - 273.15);
101
102     return maxTempInDegree;
103 }
```

JUnit test for testMax () method gives Green Bar after implementation in weatherC.

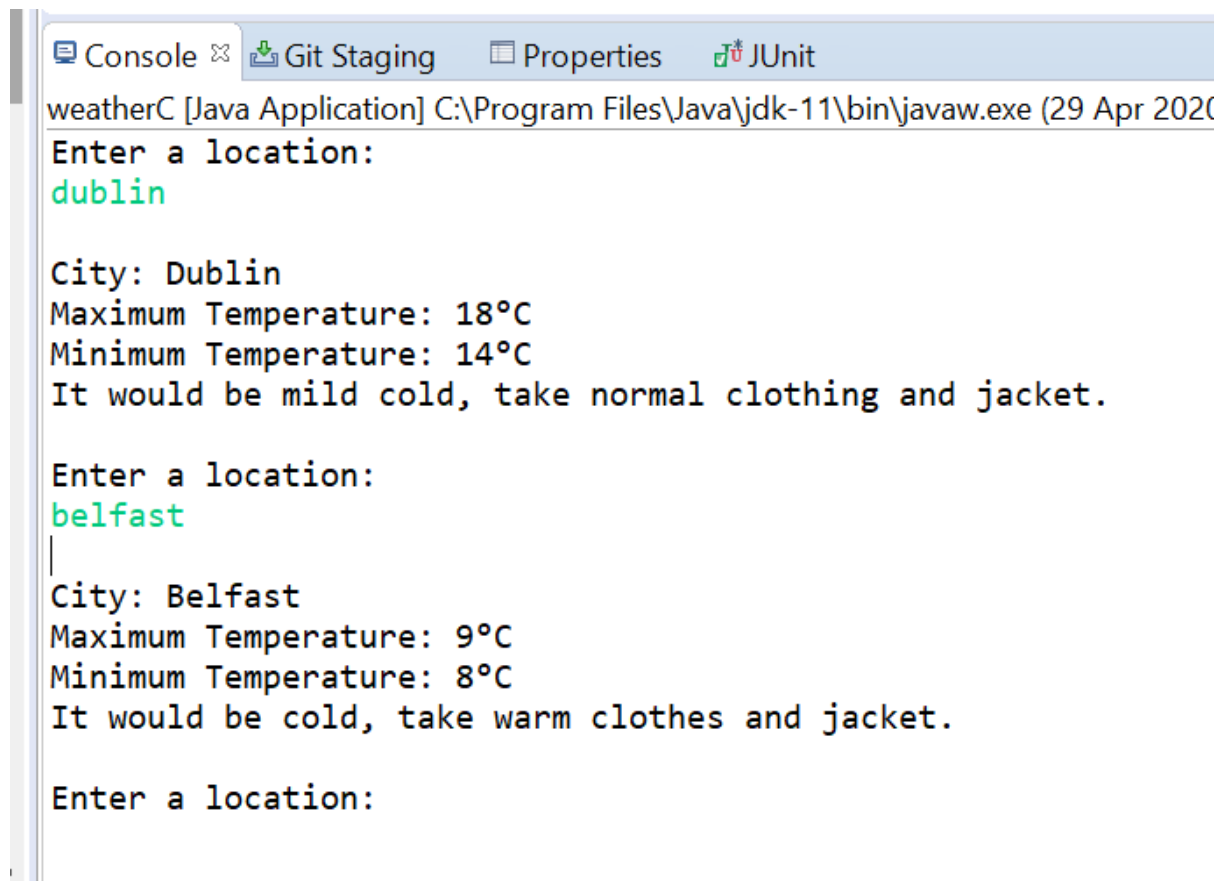


In WeatherC java file in weatherCheck () method change the output from Kelvin to degree Celsius.

```
// checking if max. temp. and min. temp. is available
if (cwd.hasMainData() && cwd.getMainData().hasTempMax() && cwd.getMainData().hasTempMin()) {
    // printing the max./min. temperature
    /*
    * System.out.println("Temperature: " + cwd.getMainData().getTempMax() + "/" +
    * cwd.getMainData().getTempMin() + "\"K");
    */
    System.out.println("Maximum Temperature: " + conversionMax(cwd.getMainData().getTempMax()) + "°C");
    System.out.println("Minimum Temperature: " + conversionMin(cwd.getMainData().getTempMin()) + "°C");

    System.out.println(cloth(cwd.getMainData().getTempMin())); // method to get clothing type
}
```

Output from WeatherC java file:



```
weatherC [Java Application] C:\Program Files\Java\jdk-11\bin\javaw.exe (29 Apr 2020)
Enter a location:
dublin

City: Dublin
Maximum Temperature: 18°C
Minimum Temperature: 14°C
It would be mild cold, take normal clothing and jacket.

Enter a location:
belfast

City: Belfast
Maximum Temperature: 9°C
Minimum Temperature: 8°C
It would be cold, take warm clothes and jacket.

Enter a location:
```

## Implementing an Array of random type of clothing

Zero () is an array of string for temperature below zero. When creating the testing for it, one string for expected and one string for actual is taken and it is tested by `assertNotEquals`. It gives a Red bar as the method does not exist yet in the `weatherC` file.

```
35 @Test
36 public void testTemperatureZero() {
37     weatherC temperatureCheck = new weatherC();
38     String Expected = temperatureCheck.zero();
39     String Actual = "Its cold";
40
41     assertEquals(Expected, Actual);
42 }
43
44
45
```

Console | Git Staging | Properties | JUnit

Finished after 0.029 seconds

Runs: 1/1 | Errors: 1 | Failures: 0

testTemperatureZero [Runner: JUnit 4] (0.000 s)

Failure Trace

java.lang.Error: Unresolved compilation problem:  
The method zero() is undefined for the type weatherC

at com.weatherCTest.testTemperatureZero(weatherCTe

Implementing an array of string in the method of `zero ()` in the `weatherC` file, a Junit green bar is obtained.

```
107 // random for temperature below 0
108 public static String zero() {
109     String[] arr={"It would be very cold, take warm clothes, jackets, scarfs and gloves.",
110                 "It would be very cold, take long warm clothes and warm jacket.",
111                 "It would be very cold, take sweater, jacket and gloves."};
112     Random r = new Random();
113     int randomNumber = r.nextInt(arr.length);
114     String coldString = arr[randomNumber];
115     return coldString;
116 }
117
```

Console | Git Staging | Properties | JUnit

Finished after 0.021 seconds

Runs: 1/1 | Errors: 0 | Failures: 0

testTemperatureZero [Runner: JUnit 4] (0.000 s)

Failure Trace

aboveZero () is an array of string for temperature above zero. When creating the testing for it, one string for expected and one string for actual is taken and it is tested by assertEquals. It gives a Red bar as the method does not exist yet in the weatherC file.

```
44 @Test
45 public void testTemperatureAboveZero() {
46     weatherC temperatureCheck = new weatherC();
47     String Expected = temperatureCheck.aboveZero();
48     String Actual = "Its not that cold";
49
50     assertEquals(Expected, Actual);
51 }
52
53
54
55
```

Runs: 1/1   Errors: 1   Failures: 0

testTemperatureAboveZero [Runner: JUnit 4] (0.000 s)

Failure Trace

java.lang.Error: Unresolved compilation problem:  
The method aboveZero() is undefined for the type weath

at com.weatherCTest.testTemperatureAboveZero(weather

After Implementing an array of string in the method of abovezero () in the weatherC file, a Junit green bar is obtained.

```
//random for temperature between 0 and 10
public static String aboveZero() {
    String[] arr={"It would be cold, take warm clothes, scarfs and gloves.",
                "It would be cold, take long warm clothes and jacket.",
                "It would be cold, take sweater, jacket and gloves."};
    Random r = new Random();
    int randomNumber = r.nextInt(arr.length);
    String coldString = arr[randomNumber];
    return coldString;
}
```

sole   Git Staging   Properties   JUnit

ed after 0.021 seconds

: 1/1   Errors: 0   Failures: 0

testTemperatureAboveZero [Runner: JUnit 4] (0.000 s)

Failure Trace

aboveTen () is an array of string for temperature above ten. When creating the testing for it, one string for expected and one string for actual is taken and it is tested by assertEquals. It gives a Red bar as the method does not exist yet in the weatherC file.

```
52
53 @Test
54 public void testTemperatureAboveTen() {
55     weatherC temperatureCheck = new weatherC();
56     String Expected = temperatureCheck.aboveTen();
57     String Actual = "Its not that cold";
58
59     assertEquals(Expected, Actual);
60 }
61
```

Failures: 0

Failure Trace

java.lang.Error: Unresolved compilation problem:

The method aboveTen() is undefined for the type weatherC

at com.weatherCTest.testTemperatureAboveTen(weatherCTest.java:56)

After Implementing an array of string in the method of aboveTen () in the weatherC file, a Junit green bar is obtained.

```
//random for temperature between 10 and 15
public static String aboveTen() {
    String[] arr={"It would be mild cold, take normal clothing and jacket.",
                 "It would be mild cold, take warm clothes and light jacket.",
                 "It would be mild cold, take warm clothes and jacket."};
    Random r = new Random();
    int randomNumber = r.nextInt(arr.length);
    String coldString = arr[randomNumber];
    return coldString;
}
```

le Git Staging Properties JUnit

after 0.027 seconds

1/1

Errors: 0

Failures: 0

tTemperatureAboveTen [Runner: JUnit 4] (0.001 s)

Failure Trace

aboveFifteen () is an array of string for temperature above fifteen. When creating the testing for it, one string for expected and one string for actual is taken and it is tested by assertNotNull. It gives a Red bar as the method does not exist yet in the weatherC file.

```

61
62 @Test
63 public void testTemperatureAboveFifteen() {
64     weatherC temperatureCheck = new weatherC();
65     String Expected = temperatureCheck.aboveFifteen();
66     String Actual = "Its warm";
67
68     assertNotNull(Expected, Actual);
69 }
70
71

```

es: 0

Failure Trace

java.lang.Error: Unresolved compilation problem:

The method aboveFifteen() is undefined for the type weatherC

at com.weatherCTest.testTemperatureAboveFifteen(weatherCTest.java:65)

After Implementing an array of string in the method of aboveFifteen () in the weatherC file, a Junit green bar is obtained.

```

//random for temperature between 15 and 18
public static String aboveFifteen() {
    String[] arr={"It would be warm, take normal clothing and a light jacket.",
        "It would be warm, take light clothes and jacket.",
        "It would be warm, take normal light clothes"};
    Random r = new Random();
    int randomNumber = r.nextInt(arr.length);
    String coldString = arr[randomNumber];
    return coldString;
}

```

Git Staging Properties JUnit

ed after 0.021 seconds

1/1

Errors: 0

Failures: 0

testTemperatureAboveFifteen [Runner: JUnit 4] (0.000 s)

Failure Trace

aboveEighteen () is an array of string for temperature above eighteen. When creating the testing for it, one string for expected and one string for actual is taken and it is tested by assertNotNull. It gives a Red bar as the method does not exist yet in the weatherC file.

```
71 @Test
72 public void testTemperatureAboveEighteen() {
73     weatherC temperatureCheck = new weatherC();
74     String Expected = temperatureCheck.aboveEighteen();
75     String Actual = "Its very warm";
76
77     assertNotNull(Expected, Actual);
78 }
79
```



#### Failure Trace

java.lang.Error: Unresolved compilation problem:  
The method aboveEighteen() is undefined for the type weatherC

at com.weatherCTest.testTemperatureAboveEighteen(weatherCTest.java:74)

After Implementing an array of string in the method of aboveEighteen () in the weatherC file, a Junit green bar is obtained.

```
//random for temperature 19
public static String aboveEighteen() {
    String[] arr={"It would be very warm, take light clothes.",
                "It would be very warm, take normal light clothes.",
                "It would be very warm, take summer type clothes"};

    Random r = new Random();
    int randomNumber = r.nextInt(arr.length);
    String coldString = arr[randomNumber];
    return coldString;
}
```

nsolve Git Staging Properties JUnit

ed after 0.022 seconds

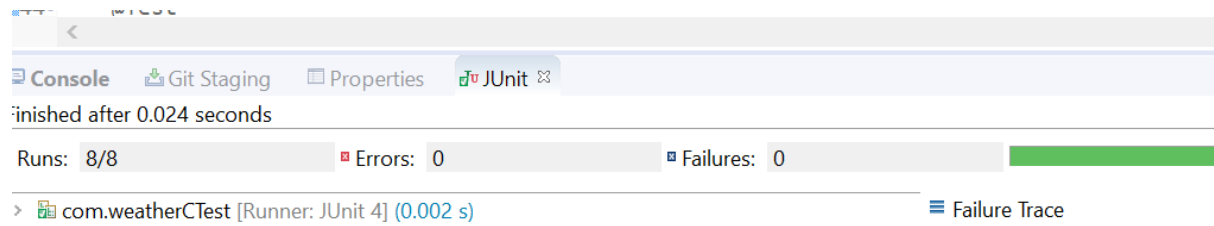
1/1 Errors: 0 Failures: 0

testTemperatureAboveEighteen [Runner: JUnit 4] (0.000 s)

Failure Trace



## Running the weatherCTest file for JUnit testing



Changes are made in the cloth () method so that it can implement the new methods.

Depending on the temperature, it goes through a selected if statement where it goes through a specific method that will choose the weather randomly.

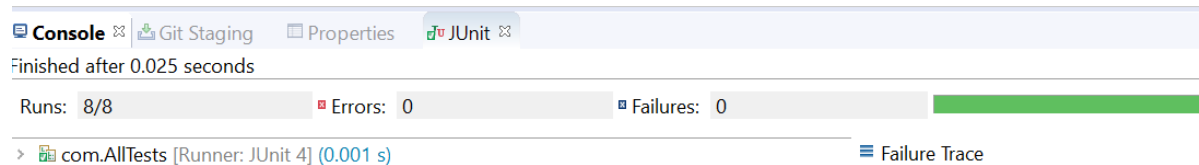
```
//cloth selection|
public static String cloth(Double tempMin) {
    String typeOfClothing = "";
    if(tempMin < 273.15) { // less than 0 degree
        typeOfClothing = zero();
    }
    else if(tempMin > 273.15 && tempMin <= 283.15 ) { //greater than 0 and less than 10
        typeOfClothing = aboveZero();
    }
    else if(tempMin > 283.15 && tempMin <= 288.15) { // greater than 10 and less than 16
        typeOfClothing = aboveTen();
    }
    else if(tempMin > 288.15 && tempMin <= 291.15) { // greater than 15 and less than 19
        typeOfClothing = aboveFifteen();
    }
    else if(tempMin >= 291.15) { //more than 18
        typeOfClothing = aboveEighteen();
    }
    return typeOfClothing;
}
```

## JUnit test suite

The test suite run all the junit method of weatherCTest file.

```
1 package com;
2
3 import org.junit.runner.RunWith;
4
5
6
7 @RunWith(Suite.class)
8 @SuiteClasses({ weatherCTest.class })
9 public class AllTests {
10
11 }
12
```

Running the test suite gives a green bar.



## Trial 1: Running the chatbot

Human : `hi`

Robot : Hi! It's delightful to see you.

Human : `how are you`

Robot : Always cheered up when I see you.

Human : `Lets check the weather`

Enter a location:

`dublin`

City: Dublin

Maximum Temperature: 26°C

Minimum Temperature: 21°C

It would be very warm, take light clothes.

Enter a location:

`cork`

City: Cork

Maximum Temperature: 12°C

Minimum Temperature: 12°C

It would be mild cold, take warm clothes and jacket.

Enter a location:

`dublin`

City: Dublin

Maximum Temperature: 26°C

Minimum Temperature: 21°C

It would be very warm, take normal light clothes.

Enter a location:

belfast

City: Belfast

Maximum Temperature: 11°C

Minimum Temperature: 10°C

It would be mild cold, take warm clothes and jacket.

Enter a location:

london

|

City: London

Maximum Temperature: 12°C

Minimum Temperature: 9°C

It would be cold, take sweater, jacket and gloves.

Human :

## DESIGN

ActionCode weather application is made up

- A java class file called the weatherC.java
- A Junit testing file called weatherCTest.java
- A Junit test suite called AllTests.java

weatherC.java contains:

- weatherCheck () method – connection to weather API and finds weather
- cloth () method – clothing selection according to temperature
- conversionMin () – conversion of minimum temperature from kelvin to degree
- conversionMax () – conversion of maximum temperature from kelvin to degree
- zero () – Array of clothes for temperature below zero it picks up clothing randomly
- aboveZero () – Randomly choose cloth from array of clothing for cold weather
- aboveTen () – Randomly choose cloth from array of clothing for mild weather
- aboveFifteen () – Randomly choose cloth from array for warm weather
- aboveEighteen () – Randomly choose cloth from array for warm weather

weatherCTest Junit test file contains:

- testClothes () method
- testMin () method
- testMax () method
- testTemperatureZero () method
- testTemperatureAboveZero () method
- testTemperatureAboveTen () method
- testTemperatureAboveFifteen () method
- testTemperatureAboveEighteen () method

AllTests.java Junit Test Suite contains:

- Join all the Junit test file by class and run it together

## RUNNING THE CHATBOT

- Add the project folder as a maven file on eclipse.
- Add the jar files
- Build the path of the jar files
- Run chatbot.java using a java application

## FUTURE ADDITION

ActionCode can be further improved by forecasting the weather for a later date. The human user enters the date that he would be going on holiday and ActionCode would go and retrieve weather information about that particular day. After obtaining the weather details ActionCode can give suggestion of clothing.

The suggestion of clothing can also be improved in near future, by adding more clothing type in each of the array for the different temperatures.

## CONCLUSION

The project took about 6 weeks to be completed. All members of the group were assigned a particular part to code. The code was put together and tested several times to ensure it is in the benefit of the chatbot. A lot more changes can be implemented to ActionCode to make it better and improve all its weaknesses.

At this instance, ActionCode can find the weather of a particular place by the name of the place. The minimum and maximum temperature are given in degree. A random selection of cloth is given by ActionCode from an array depending on minimum temperature as well as a suggestion of clothing will be given with temperature accordingly.